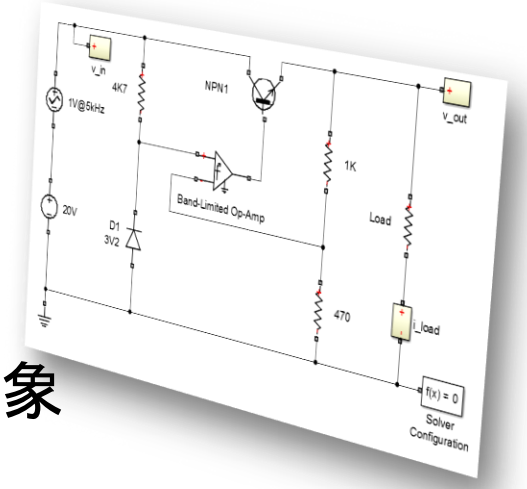
モデルを活用した
パワーエレクトロニクス制御開発：
実装/試験編

MathWorks Japan
アプリケーションエンジニアリング部
アプリケーションエンジニア
新井　克明

パワーエレクトロニクス制御と広がる活用分野



- パワーエレクトロニクス制御
  - 電力の有効・安全利用のために
    電流・電圧を所望の形に変換/制御
  - 直流/交流/電圧/電流/周波数/位相が制御対象

- 活用分野


電源/UPS


太陽光発電/
パワーコンディショナ


EV/HEV

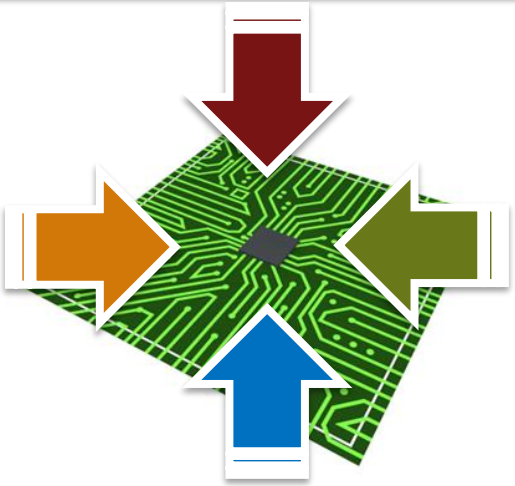

HEMS

パワーエレクトロニクスに対するニーズ

- 限られた開発リソース/機能的制約下での付加価値の創出

■高効率化
- 環境規制対策
  （e.g. 省エネ法 / 国際エネルギースタープログラム）
- 動作時/待機時の省電力の両立

■高品質化
- 低リプル
- デバイス/負荷保護
- 瞬停/熱/EMC対策

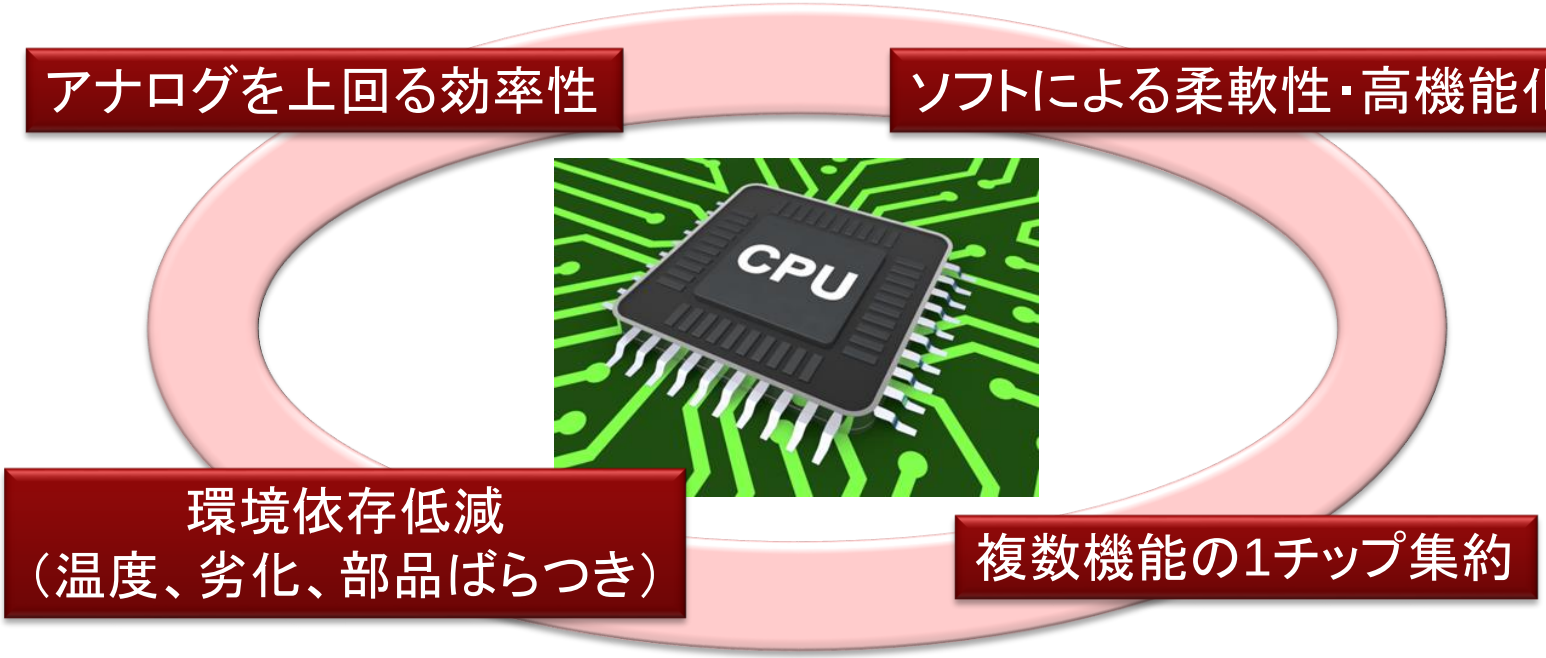


■小型化・軽量化
- 携帯機器向け電源
- 高密度サーバ用電源
- 車載コンバータ・インバータ

■高機能化
- インテリジェントな電力管理（創・蓄・省エネ）
- 通信による他システムとの連携

パワーエレクトロニクスとデジタル制御

- デジタル制御がパワーエレクトロニクスで徐々に浸透中



制御性能は制御手法/アルゴリズムの良し悪しに依存

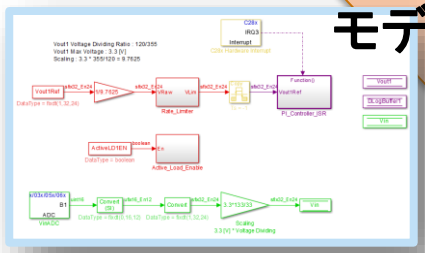
モデルを活用したパワエレシステム開発

- ✓ シミュレーション: 試作前に様々な検証が可能、手戻り回避
- ✓ 実装/試験: 様々なハードウェアや試験環境に実装可能

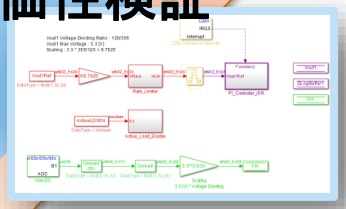
制御ソフト＋回路
シミュレーション検証

制御ソフト実装
実回路検証

コード実装に向けた
モデル詳細設計



モデル＆コード
等価性検証



自動コード生成

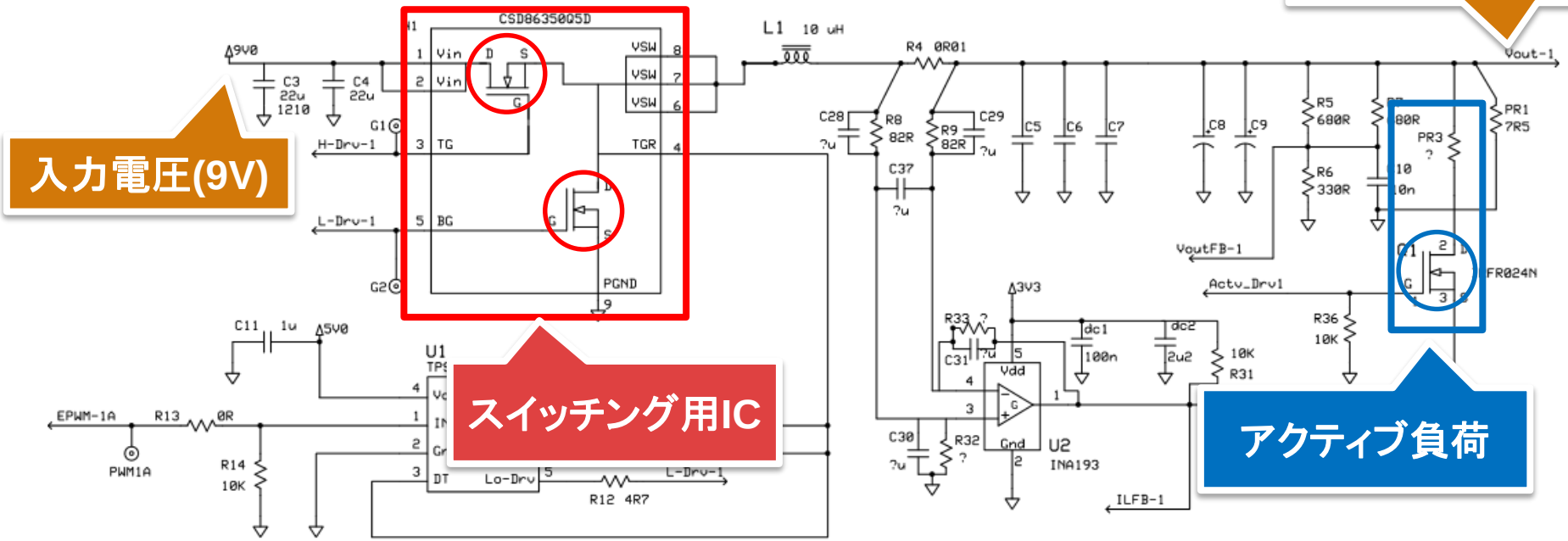
本講演でご紹介する内容

アジェンダ

- ソフトウェアへのアルゴリズム実装
- ハードウェアへのアルゴリズム実装
- 実機試験/他ツール連携ソリューション
- まとめ

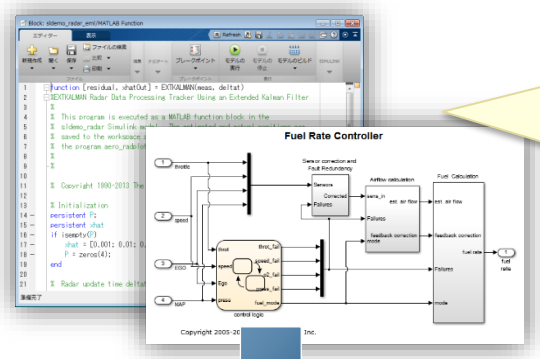
実装のターゲット

- Texas Instruments C2000
  DC-DCコンバータトレーニングキット
  - 搭載マイコン: F28035
    - 32bit CPU
    - 駆動周波数: 60MHz



モデル＋Cコード自動生成ソリューション

モデル



Simulink® / Stateflow®

- モデル作成・シミュレーション基本環境

Fixed-Point Designer™

- 固定小数点設計

自動生成
コード

```
if (reset) {
  y = 0;
} else {
  y += k * u;
}
```

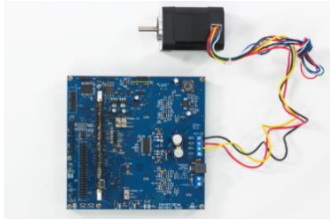
C++

Embedded Coder®

- モデルから効率性・可読性に優れた C/C++コードを自動生成

※ライセンス上、MATLAB Coder, Simulink Coderも必要

既存コードと
組み合わせて利用



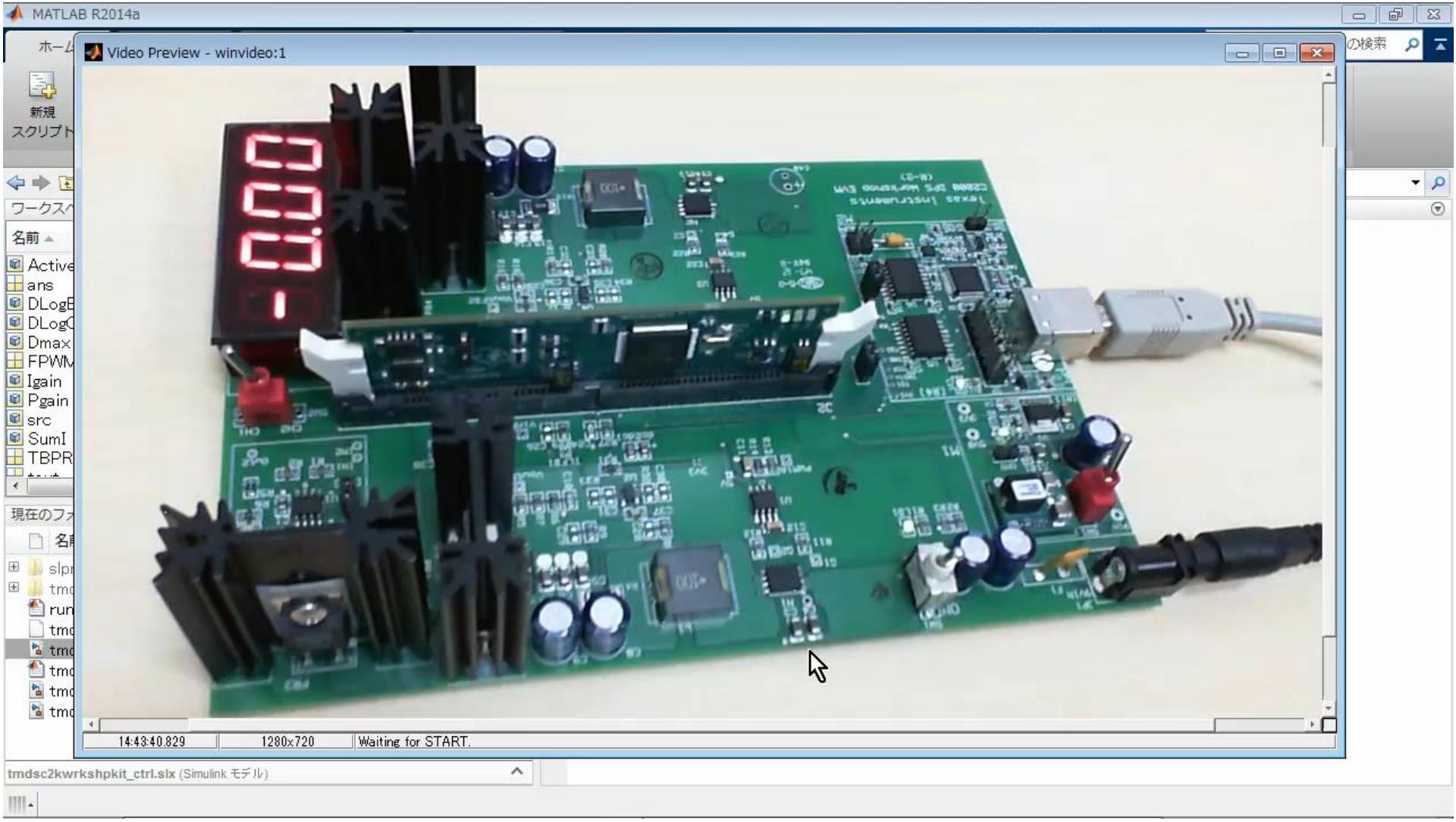
MCU/DSPに実装して
実験・量産試作

モデルをDLL化して
PC環境で利用

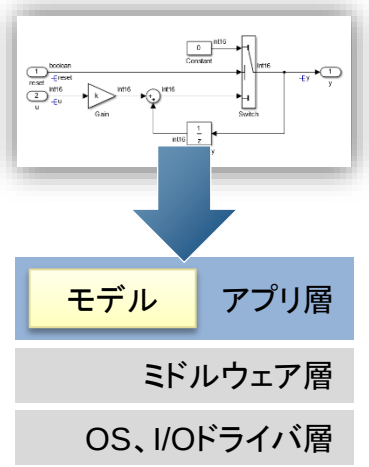
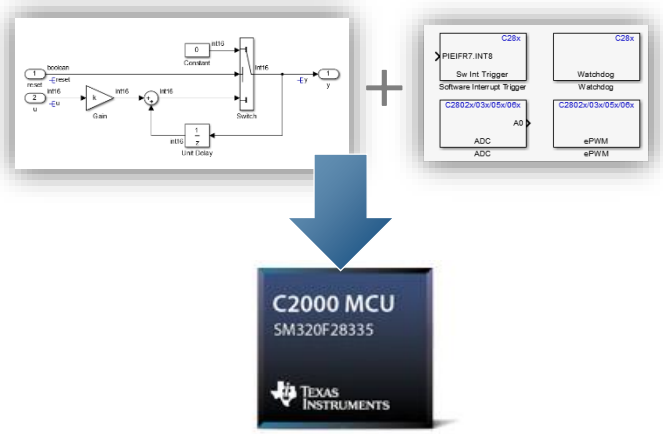
モデル生成コードの利用は無料

モデルによるソフトウェア実装ダイジェスト

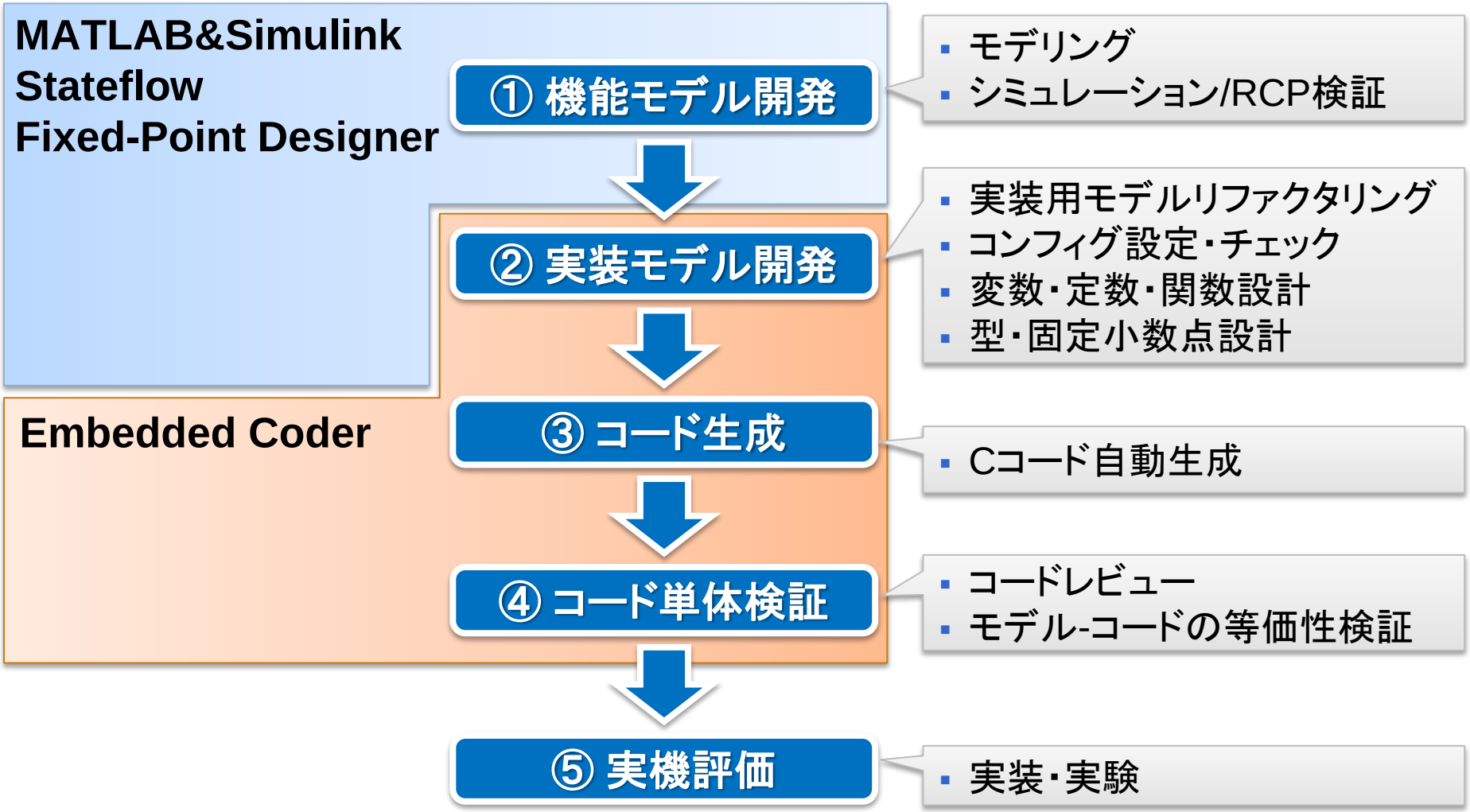
MOVIE



Embedded Coder モデル生成コードの利用タイプ

	ソフトコンポーネント型（基本）	モデルビルド型（応用）
利用イメージ	アプリの一部をモデル化、生成されたコードを既存ソフトに組み込む 	MCU/DSP用ブロックライブラリ利用、モデルを直接実装する 
主な用途	既存ソフトとの差分開発 HW/OS非依存アプリの開発	MCU/DSPを用いた制御実験 （On Target RCP）
特徴	標準ブロックのみ使用	専用ブロックライブラリを一部使用
I / O	グローバル変数 / 引数	AD/DA, PWM, CAN等のI/Oブロック
注意点	ビルド・結合は手動で行う必要アリ	対応MCU/DSP限定、HW知識も必要
対応 MCU/DSP	生成コードはANSI/ISO-C準拠なので ほぼ全てのMCU/DSPに対応可能	TI C2000, STMicro STM32F4 等 （MathWorksまたはベンダから提供）

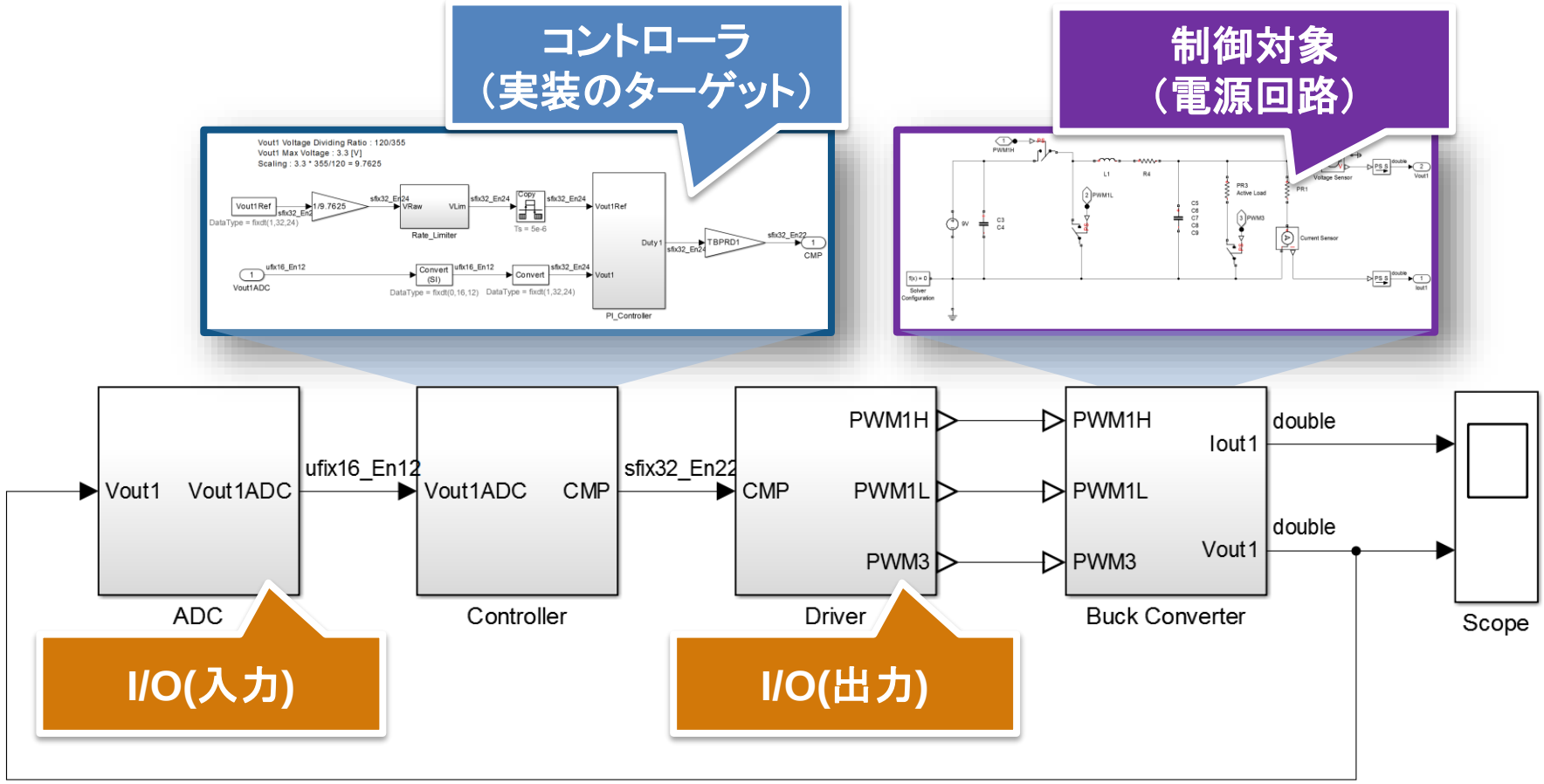
ソフトウェア実装に向けたワークフロー
Embedded Coderによるコード生成＆検証(モデルビルド型)



DC-DCバックコンバータシステムモデル

① 機能モデル開発

- コントローラ＋電源回路をモデルで表現
- 制御アルゴリズムの性能をシミュレーションで評価

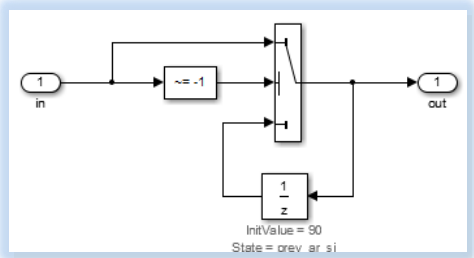


※電源回路はSimElectronicsを使用

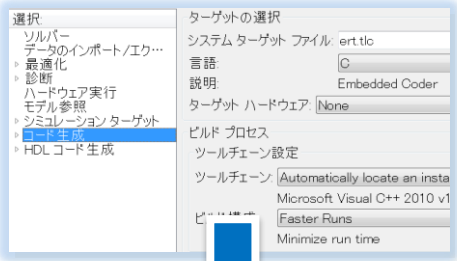
自動コード生成に必要な情報

② 実装モデル開発

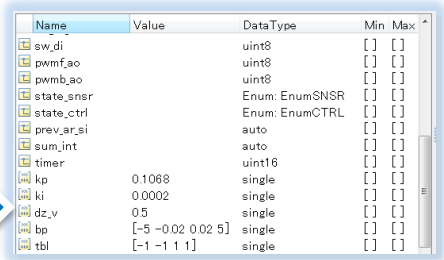
1 モデル
アルゴリズム



2 コンフィギュレーション
コード生成設定



3 データオブジェクト
データディクショナリ
変数・定数属性



Name	Value	DataType	Min	Max
sw_di		uint8	[]	[]
pwmf_ao		uint8	[]	[]
pwmb_ao		uint8	[]	[]
state_snsr		Enum: EnumSNSR	[]	[]
state_ctrl		Enum: EnumCTRL	[]	[]
prev_ar_si		auto	[]	[]
sum_int		auto	[]	[]
timer		uint16	[]	[]
kp	0.1068	single	[]	[]
ki	0.0002	single	[]	[]
dz_v	0.5	single	[]	[]
bp	[-5 -0.02 0.02 5]	single	[]	[]
tbl	[-1 -1 1 1]	single	[]	[]

Embedded Coder

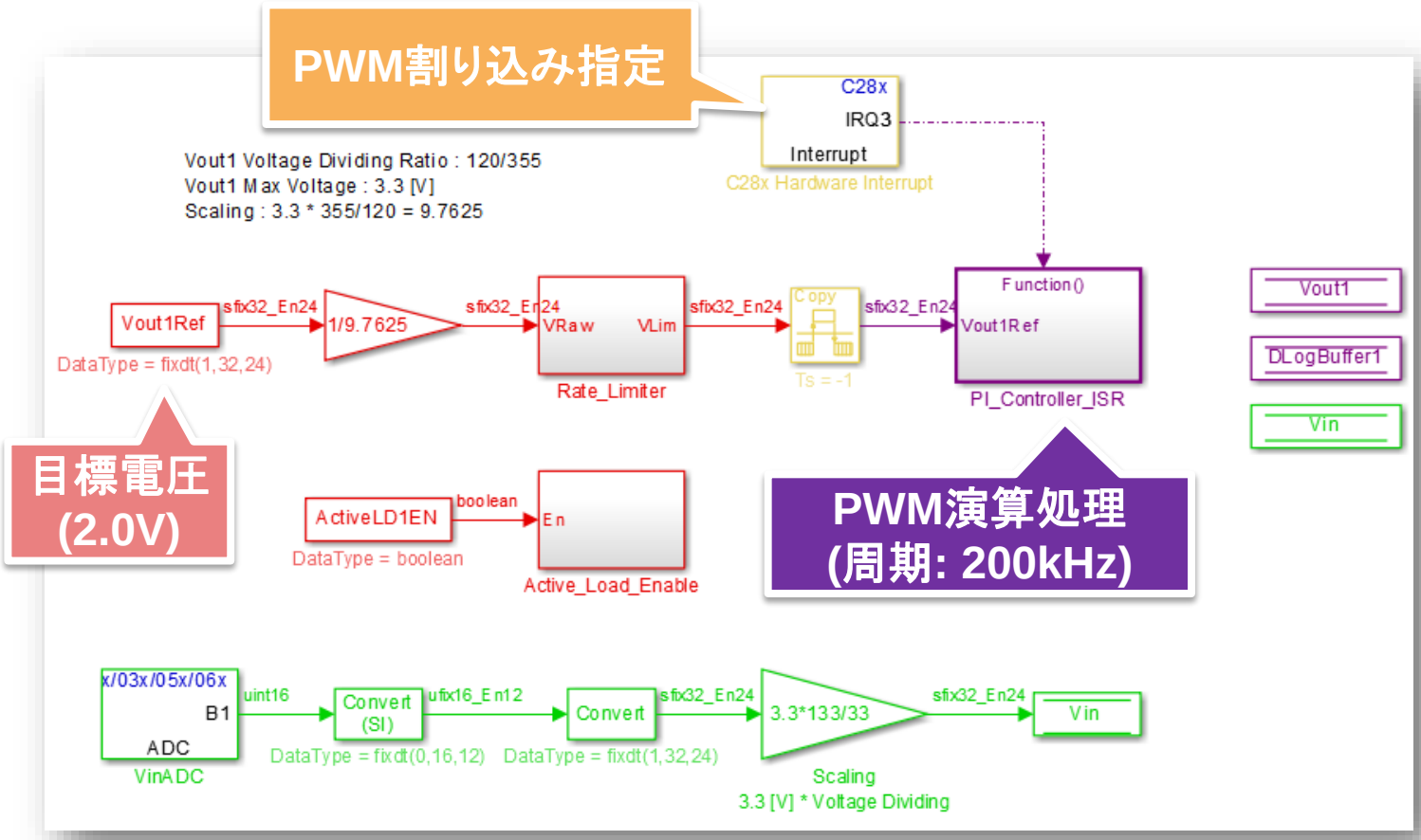
1 アルゴリズムの設計
2 コード生成関連の設定
3 変数・定数の属性定義
で自動生成されるコードが
形成されます

```
/* Hardware Interrupt Block: '<Root>/C28x Hardware Interrupt' */
void isr_int3piel_task_fcn(void)
{
  /* Call the system: <Root>/PI Controller ISR */
  {
    /* Output and update for function-call system: '<Root>/PI Controlle
    {
      /* local block i/o variables */
      int32_T rtb_IQNxIQN;
      int32_T rtb_Gain;
      int32_T rtb_Saturation;

      /* S-Function (c2802xadc): '<S2>/Vout1ADC' */
      {
        /*  Internal Reference Voltage : Fixed scale 0 to 3.3 V range.  */
        /*  External Reference Voltage : Allowable ranges of VREFHI(ADCINA0) = 3.3 and VREFLO(tied to ground) = 0
        tmdsc2kwrkshpkit_ctrl_DW.Vout1ADC = (AdcResult.ADCRESULT1);
      }

      /* DataTypeConversion: '<S2>/Data Type Conversion' */
      rtb_IQNxIQN = (int32_T)tmdsc2kwrkshpkit_ctrl_DW.Vout1ADC << 12U;

      /* Gain: '<S4>/Scaling 3.3 [V] * Voltage Dividing' */
      rtb_Gain = c28x_mul_s32_s32_s32_sr(39987L, rtb_IQNxIQN, 12L);

      /* Assignment: '<S4>/Assignment' incorporates:
       *  DataTypeConversion: '<S4>/Data Type Conversion'
       *  UnitDelay: '<S4>/Unit Delay'
       */
      DLogBuffer1[(int16_T)DLogCnt] = (int16_T)(rtb_Gain >> 13U);

      /* DataStoreWrite: '<S4>/Data Store Write1' */
      Vout1 = rtb_Gain;
```

自動生成
Cコード

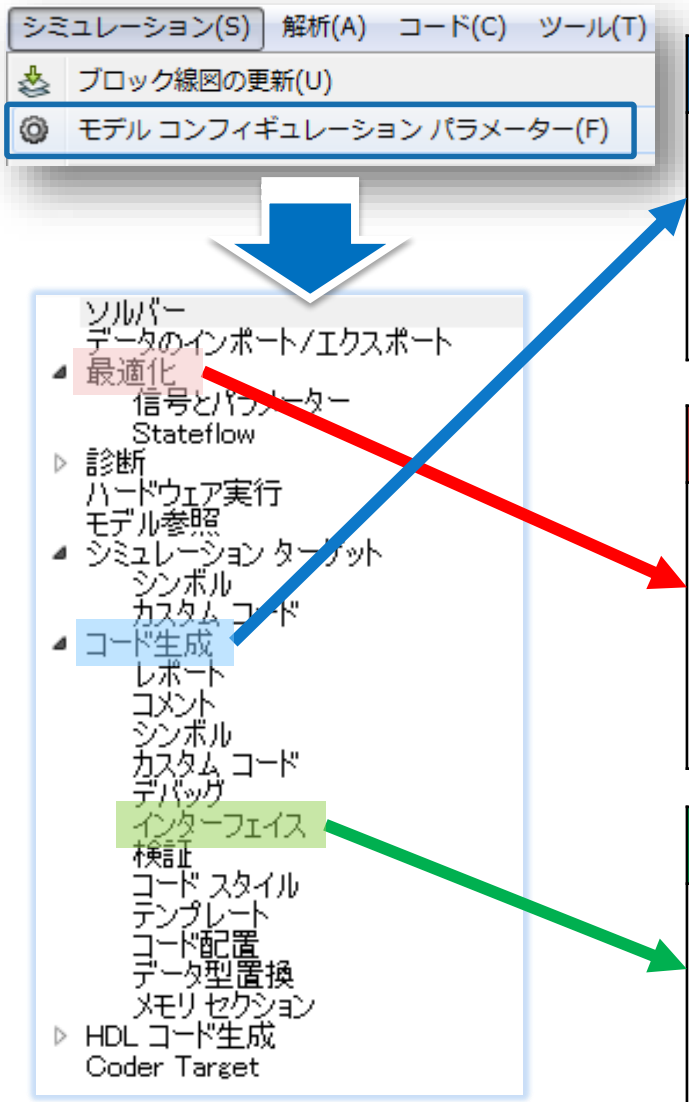
1 DC-DCバックコンバータ実装モデル

- システムモデルから実装対象（コントローラ）を取り出したモデル
- コントローラーの入出力などをI/Oブロックに置き換え

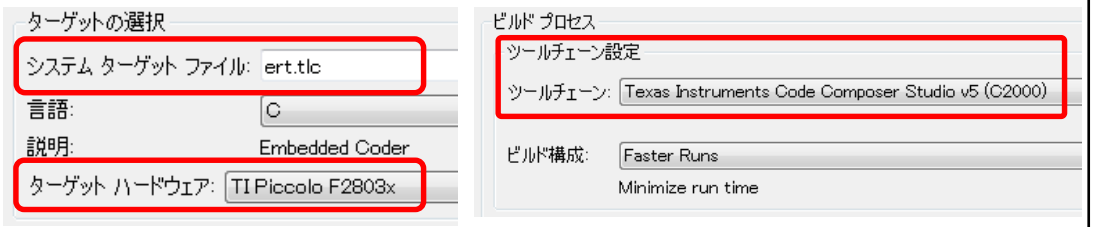


2 コード生成のコンフィギュレーション

② 実装モデル開発



ターゲットおよびビルドツールチェーンの指定



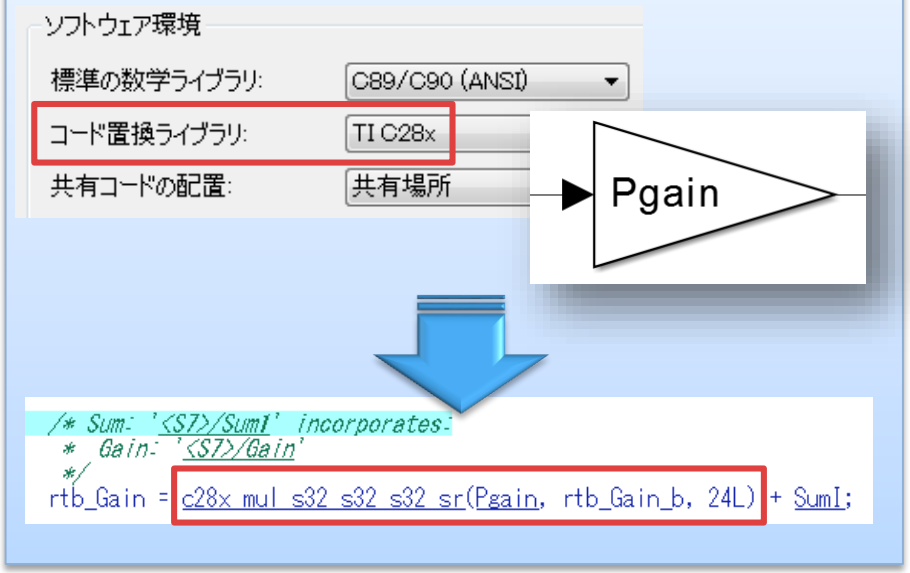
生成コードの最適化設定

コード置換の設定

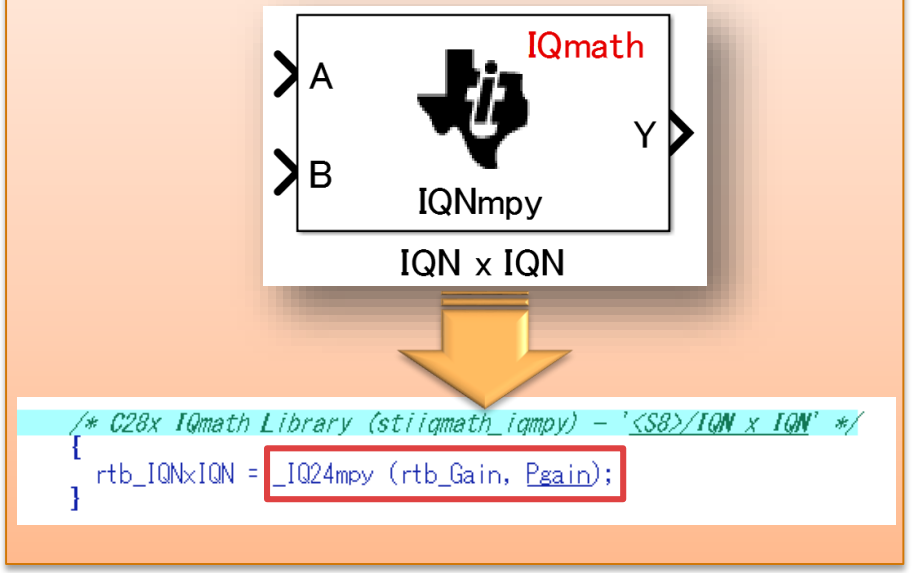
TI C2000モデルビルド型でのコード置換

- コード置換設定＆専用ブロックで生成コードをカスタマイズ

■設定によるコードの置き換え



■専用ブロックによるコードの置き換え



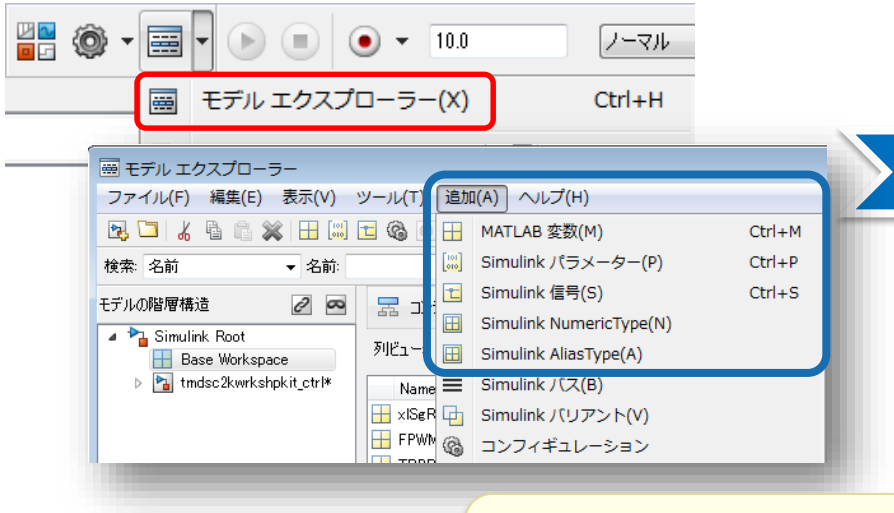
- ✓ TI C2000専用関数で演算処理の高速化を図れます

処理スピードはPILSによる実行プロファイルレポートで確認

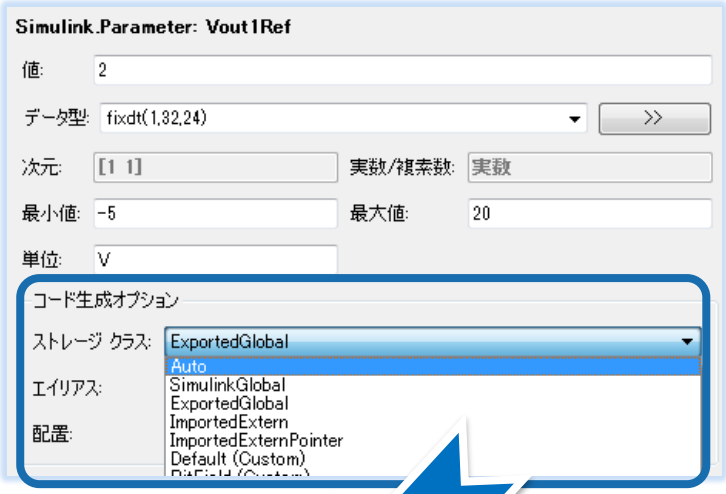
3 変数・定数（RAM/ROM）設計

コード生成時の変数の取り扱いをカスタマイズ

◆ 変数の定義

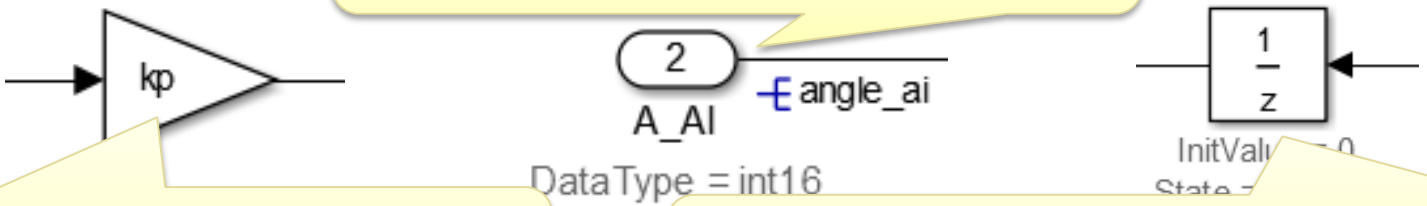


◆ データ型/宣言方法の指定



信号をグローバル変数として定義

```
int16_T angle_ai;
```



パラメータをマクロで定義

```
#define kp 0.1068F;
```

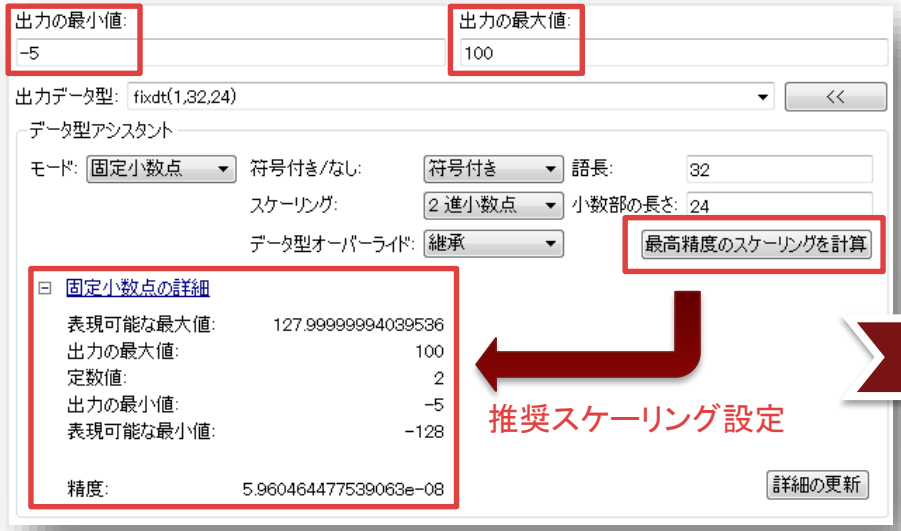
状態（バッファ変数）にstatic修飾子の付与

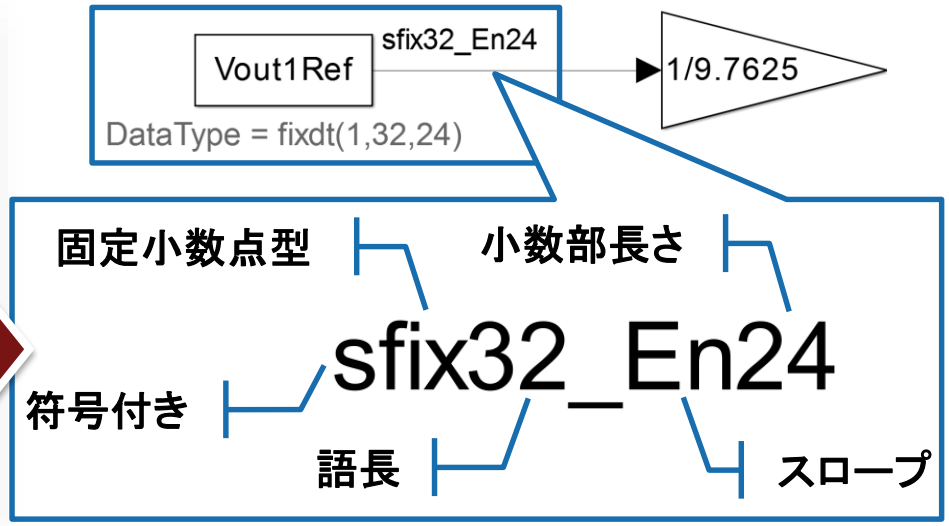
```
static real32_T sum_int;
```

3 Simulinkモデルの固定小数点設計

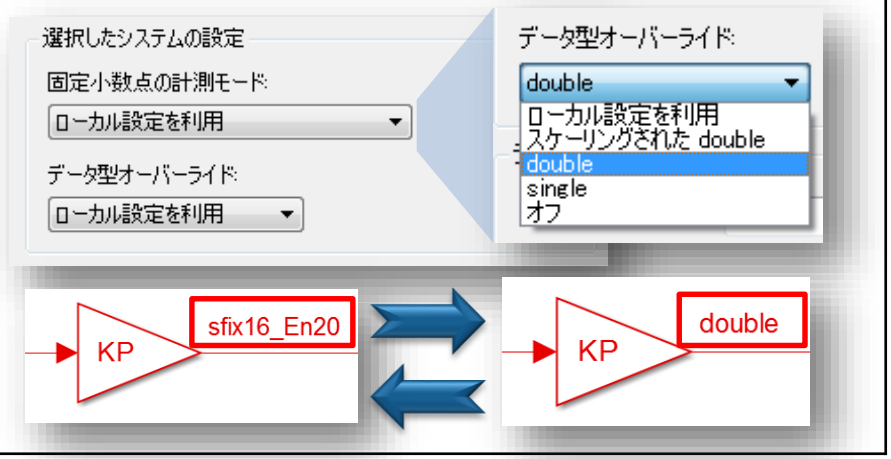
② 実装モデル開発

Fixed-Point Designerが固定小数点設計を強力サポート



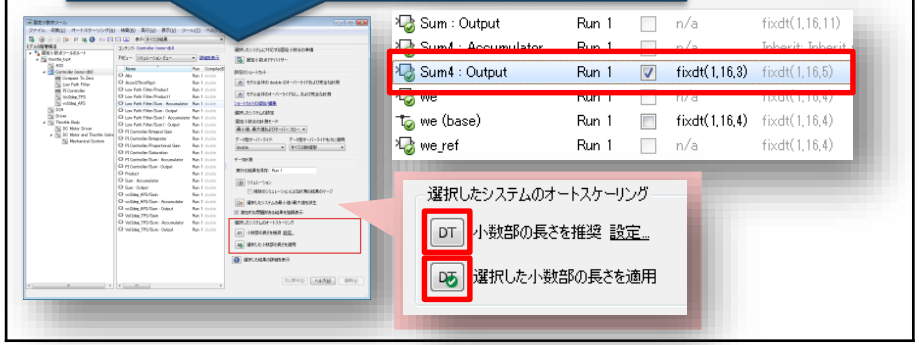


モデル全体のデータ型を一括オーバーライド



各ブロックの語調・スケーリングを一括自動調整

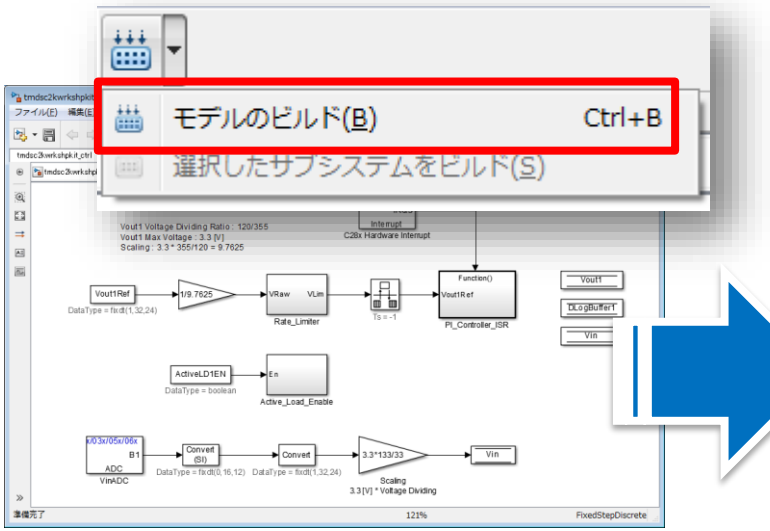
専用GUI上でモデル内のブロックの
最適な小数点設定を自動的に推定



コード生成＆コードレビュー

③ コード生成

- 諸設定が済んだ後はビルドボタン一つでコードを生成できます



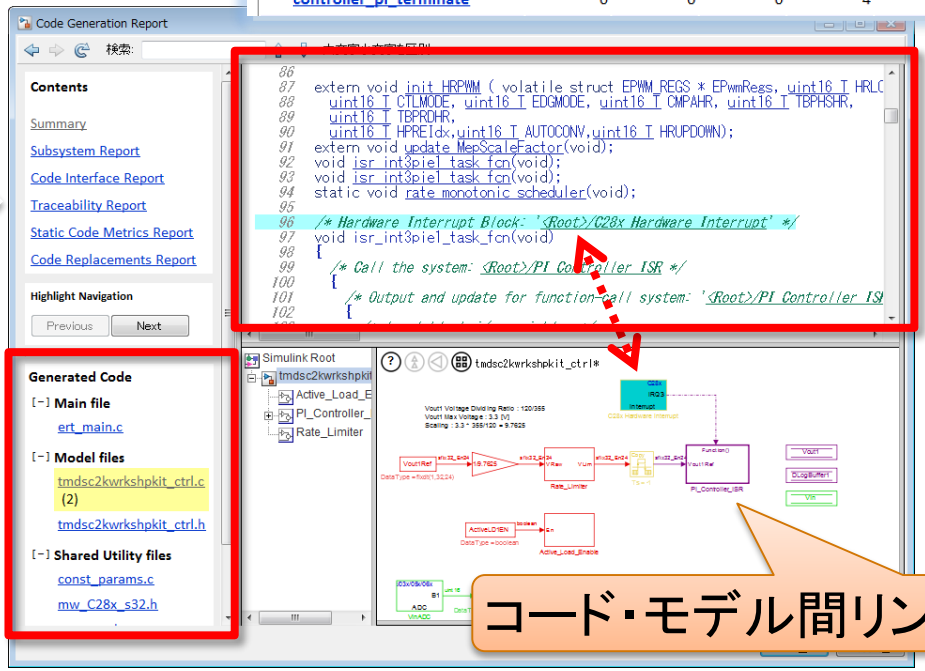
コードメトリクス
（行数、複雑性、RAMサイズ等）

3. Function Informa

View function metrics
estimated stack size of
subroutines that the function calls.

View:Call Tree | Table

Function Name	Accumulated Stack Size (bytes)	Self Stack Size (bytes)	Lines of Code	Lines	Complexity
[+] controller_pi_step	28	14	63	144	12
[+] controller_pi_initialize	22	0	10	26	1
controller_pi_terminate	0	0	0	4	1



生成ファイルリスト

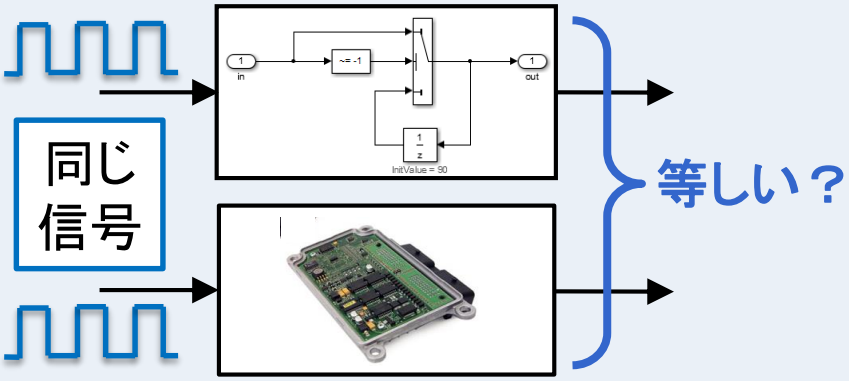
コード・モデル間リンク

※モデルビューが無い場合、モデルと直接リンクされます

等価性検証&パフォーマンステスト

- 処理系依存動作やコード生成設定不備による問題をチェック

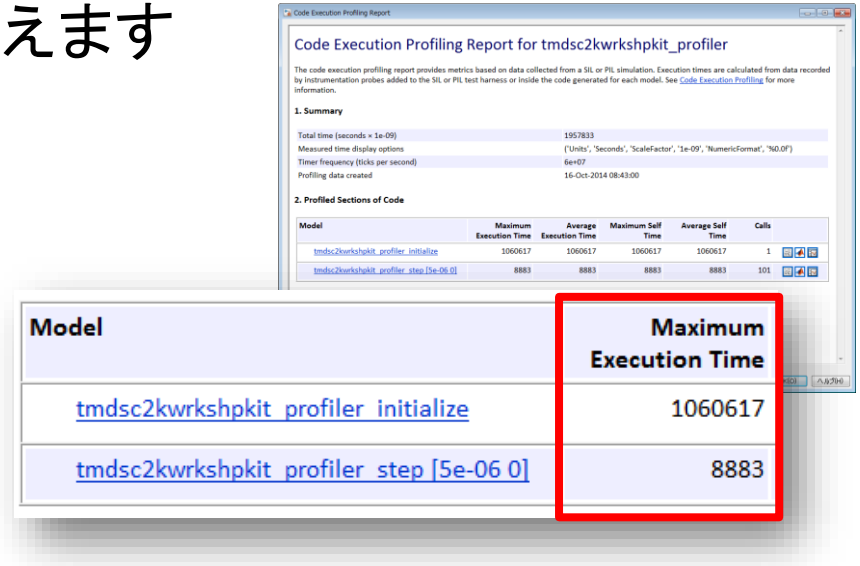
PILS: Processor In the Loop Simulation
マイコン/マイコンシミュレータ上で
生成コードを動作（ターゲット検証）



- ✓ PILSの場合演算負荷の測定が行えます

使用ブロック / 最適化設定	演算時間
TI専用ブロック	2.317μsec
通常ブロック(コード置換あり)	3.517μsec
通常ブロック	8.883μsec

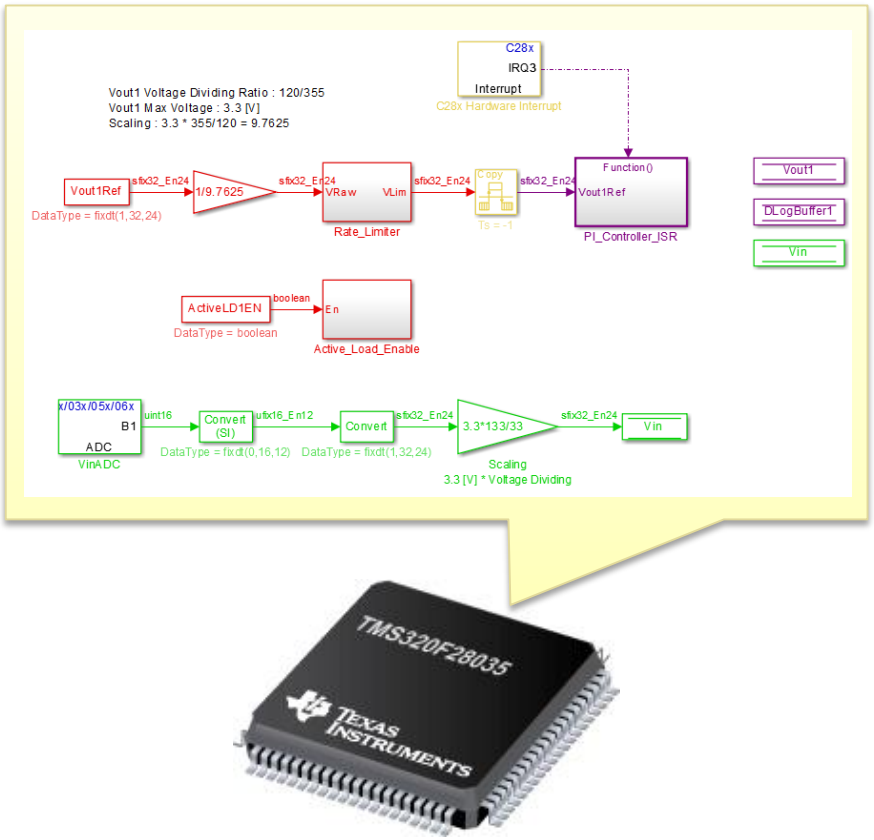
コード置換された処理なら
PWM一周期(5μsec)内で演算可能！

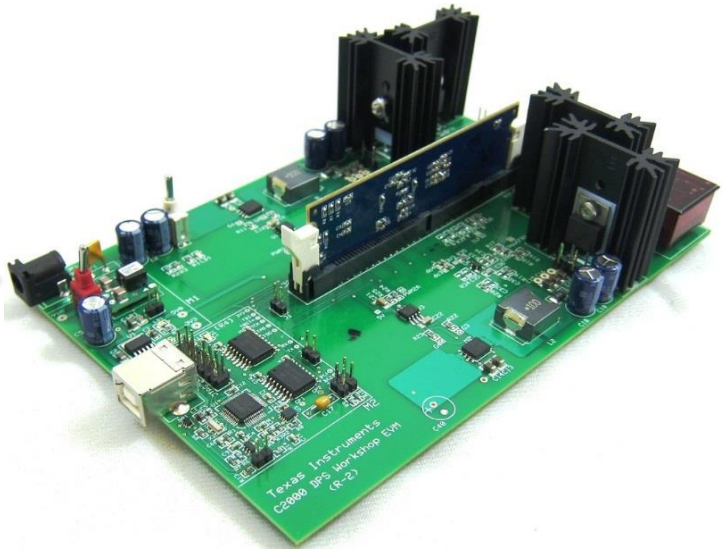

Code Execution Profiling Report for tmdsc2kwrkshpkit_profiler

Model	Maximum Execution Time
tmdsc2kwrkshpkit_profiler_initialize	1060617
tmdsc2kwrkshpkit_profiler_step [5e-06 0]	8883

実機試験

- 実行ファイルはビルド時にMCUへ自動的にダウンロード
- モデルのデザイン完了後、スグに実機試験へ移れます



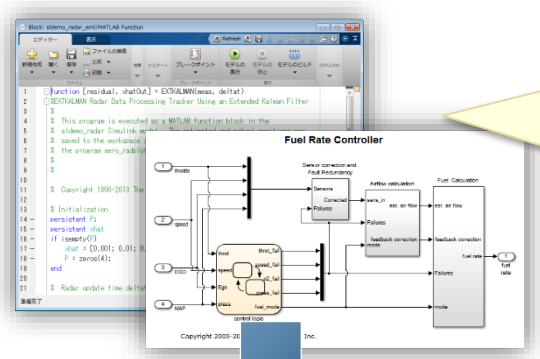


アジェンダ

- ソフトウェアへのアルゴリズム実装
- ハードウェアへのアルゴリズム実装
- 実機試験/他ツール連携ソリューション
- まとめ

モデル＋HDLコード自動生成ソリューション

モデル



自動生成
コード

```
always @(posedge CLK) begin
    if( QL==4'd9 )
        QL <= 4'd0;
    else
        QL <= QL + 1'b1;
    end
end
```

VHDL,
Verilog

既存コードと
組み合わせて利用

FPGA/ASICに実装して
実験・量産試作

Simulink® / Stateflow®

- モデル作成・シミュレーション基本環境

Fixed-Point Designer™

- 固定小数点設計

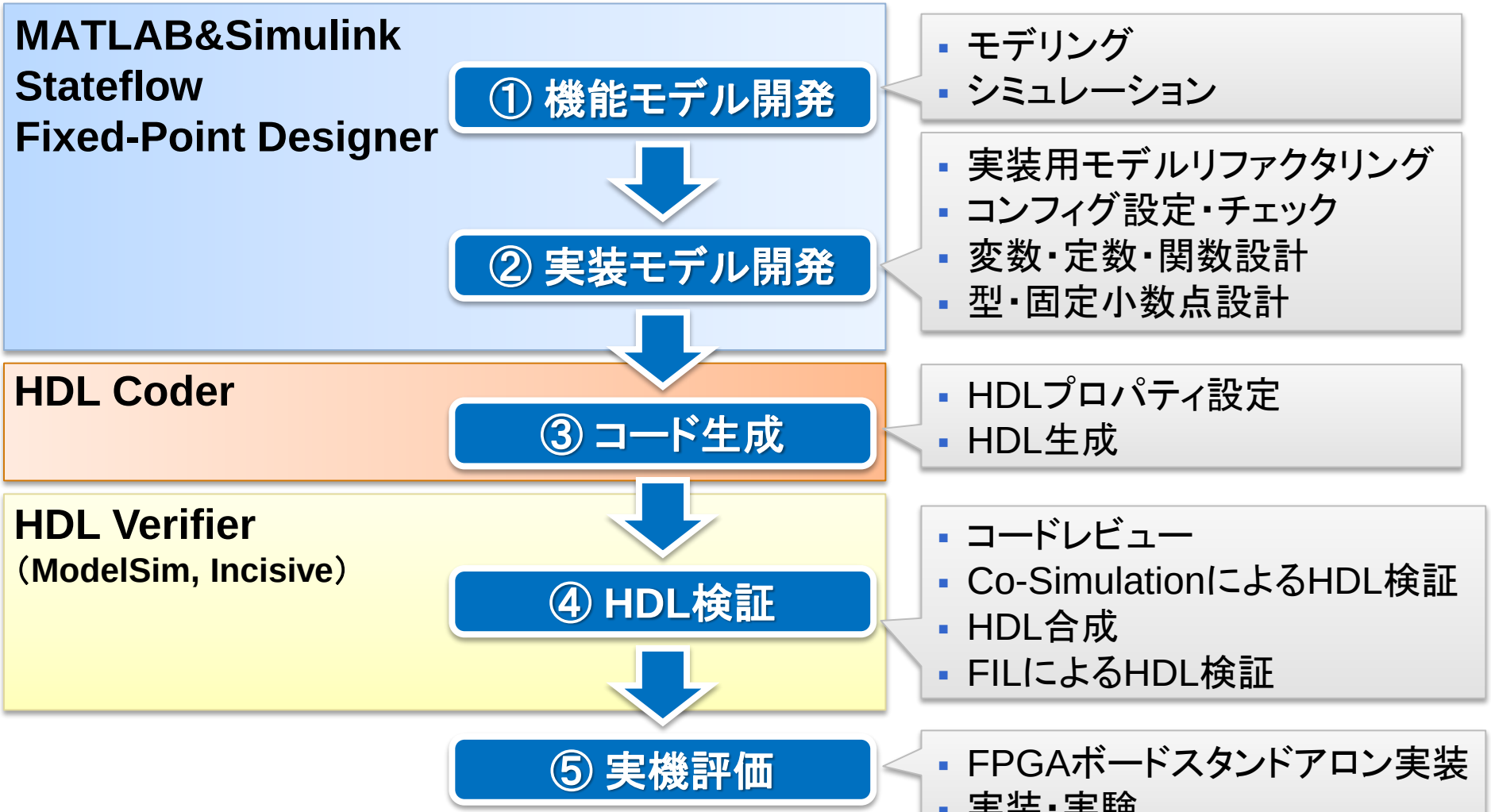
HDL Coder™

- HDLコード(VHDL/Verilog)の生成

HDL Verifier™

- HDLコードの検証機能の提供

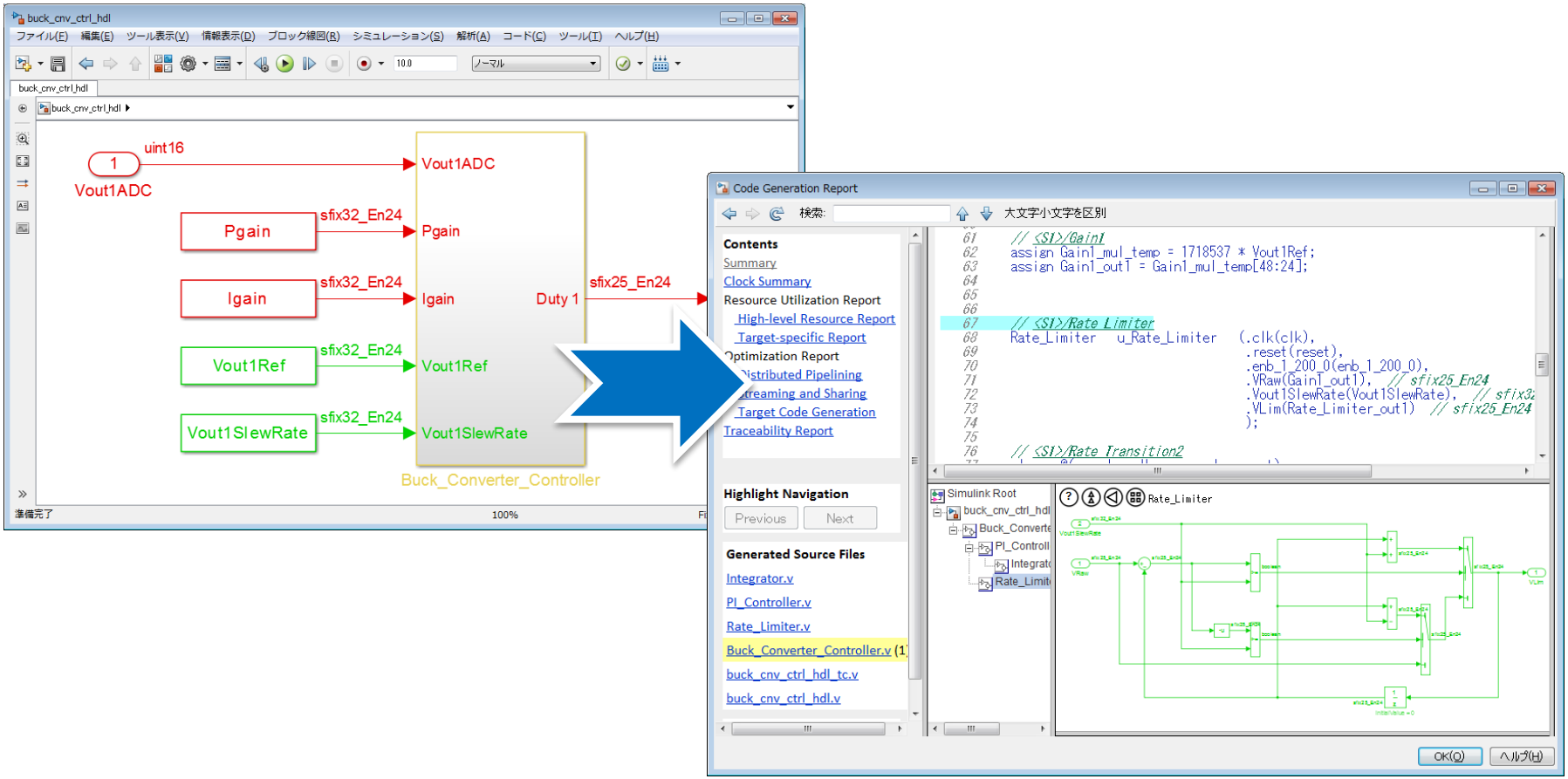
ハードウェア実装に向けたワークフロー
HDL Coder / HDL Verifierによるコード生成＆検証



FIL: FPGA In the Loop

HDL Coderによるコード生成

- Cコード生成同様、モデルから直接HDLコードを生成
- レポートのトレーサビリティ機能でコードレビューもスムーズ



ハードウェア実装ソリューションの主な機能

◆ HDL Coder

ターゲット非依存のVHDL/Verilog生成

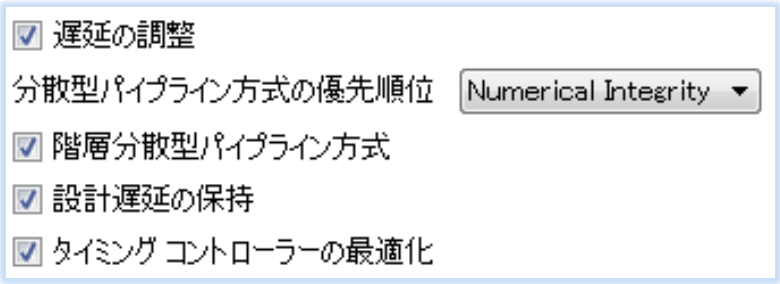
- ✓ VHDL(IEEE1076-1993)
- ✓ Verilog(IEEE1364-2001)

```
-- <S14>/Gain
Gain_mul_temp <= 5094059 * In1_signed;
Gain_out1 <= Gain_mul_temp(46 DOWNTO 23);
```

```
// <S14>/Gain
assign Gain_mul_temp = 5094059 * In1;
assign Gain_out1 = Gain_mul_temp[46:23];
```

面積と速度間のトレードオフの調整

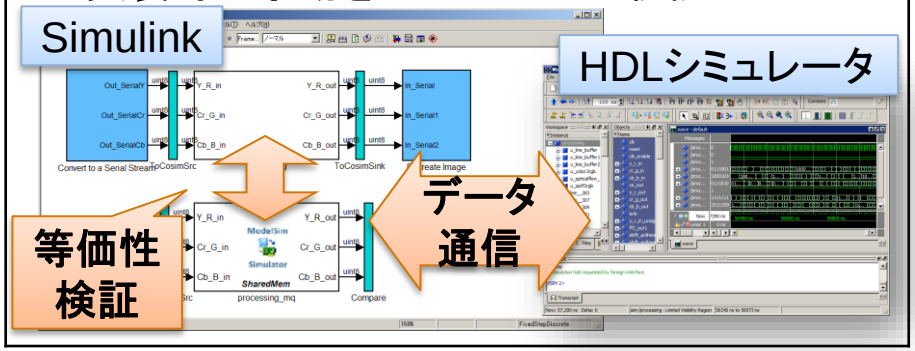
- ✓ パイプラインの追加とリソース共有の設定
- ✓ クリティカルパスの表示



◆ HDL Verifier

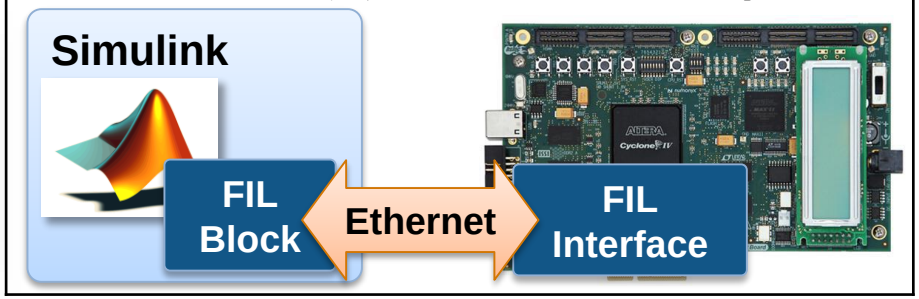
HDL開発ツールとの協調シミュレーション

- ✓ 実装時の挙動をシミュレータで検証



FPGA in the Loopによる検証

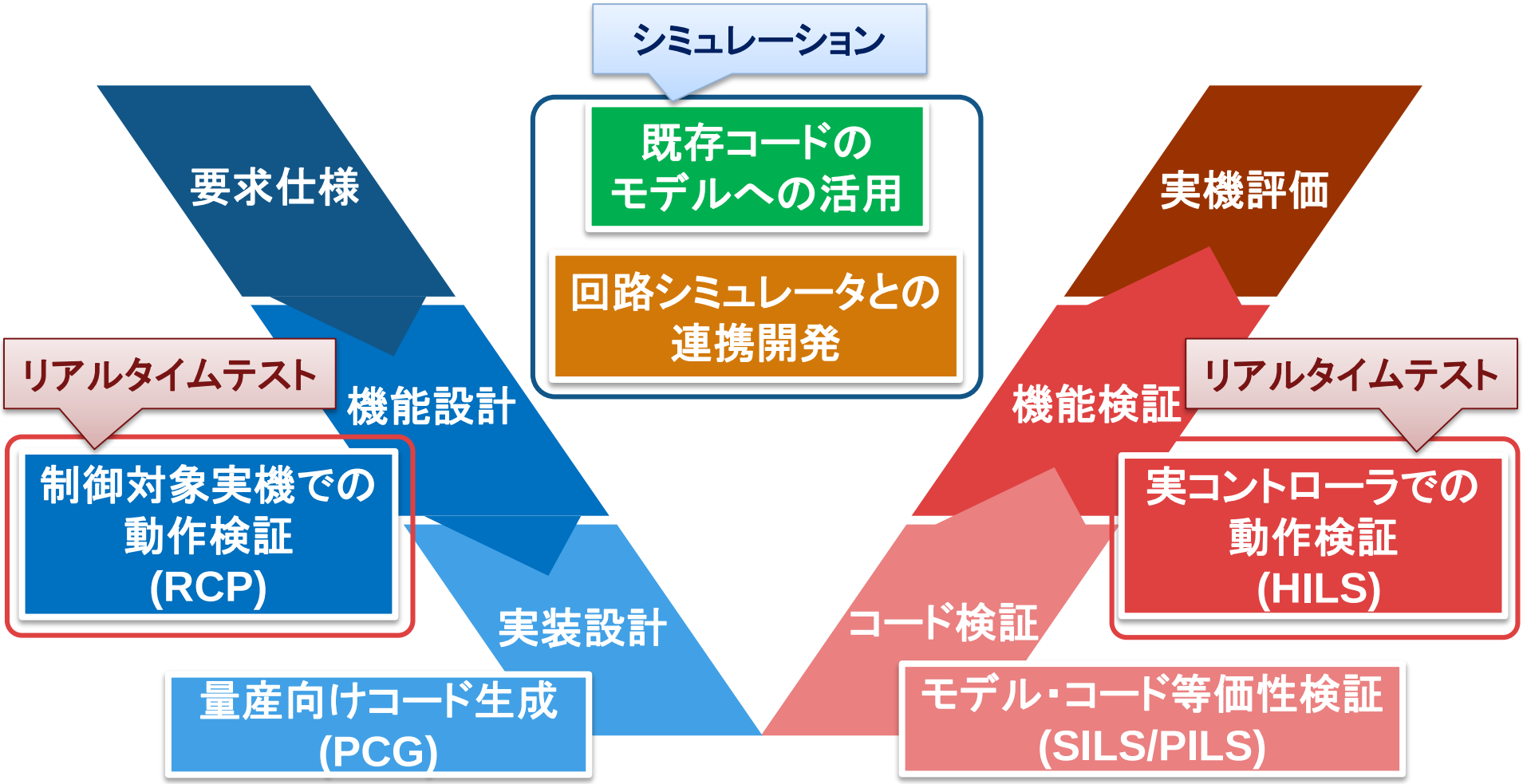
- ✓ FPGAボードによる高速プロトタイプ検証
- ✓ Xilinx・Altera汎用/カスタムボードに対応



アジェンダ

- ソフトウェアへのアルゴリズム実装
- ハードウェアへのアルゴリズム実装
- 実機試験/他ツール連携ソリューション
- まとめ

実機試験/他ツール連携ソリューション



***RCP**: **R**apid **C**ontrol **P**rototyping*

***PCG**: **P**rodution **C**ode **G**eneration*

***SILS**: **S**imulation **I**n the **L**oop **S**imulation*

***PILS**: **P**rocessor **I**n the **L**oop **S**imulation*

***HILS**: **H**ardware **I**n the **L**oop **S**imulation*

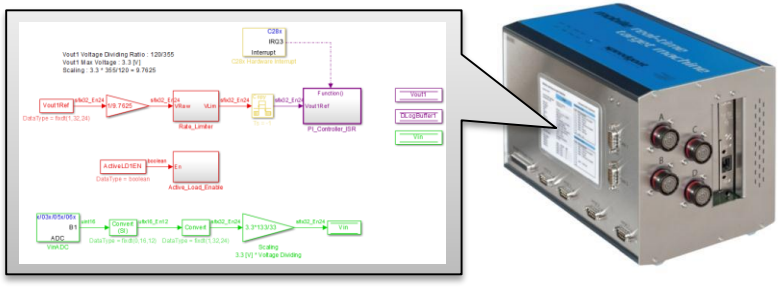
リアルタイムテストソリューション

RCP (Rapid Control Prototyping)

✓ 実装基板完成まで待てない
✓ モデルでスグに実機を動かしたい

マイコン向け設定無しで制御モデルを即座に実行 ＆ 実制御対象でテスト

コントローラーモデル



制御対象
実機

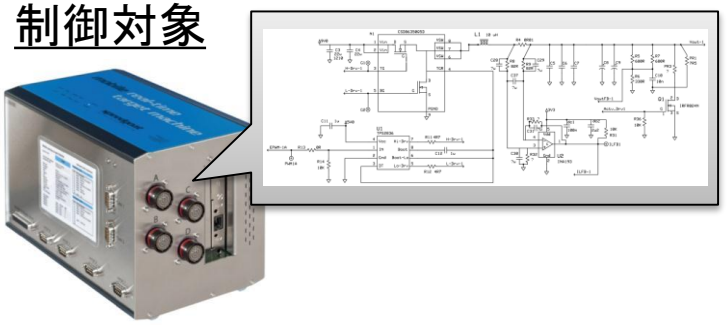
HILS (Hardware In the Loop Simulation)

✓ 本番の実機試験に向けた事前チェック
✓ 再現が難しいテストをしたい

制御対象を模擬したシミュレーション環境でコントローラ実機を詳細テスト

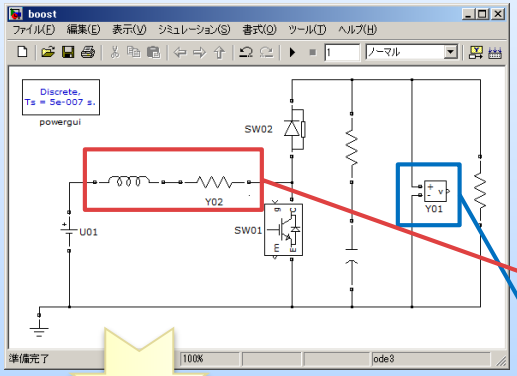
コントローラ
実機

制御対象

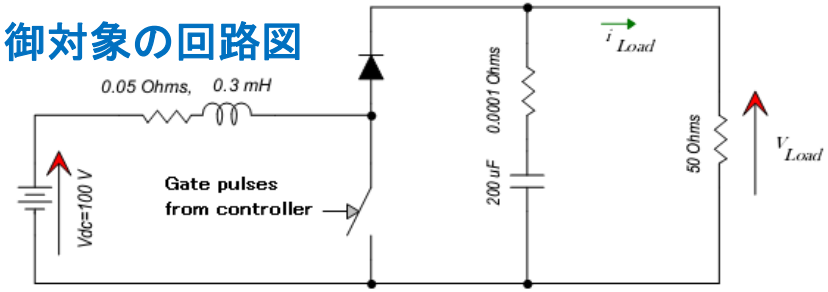


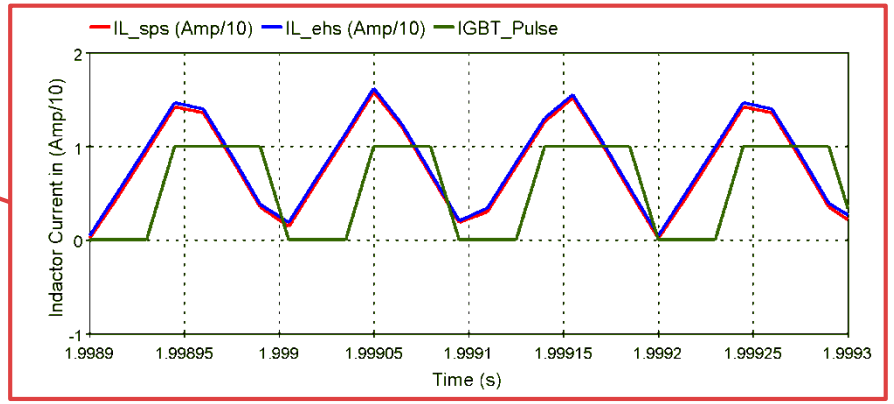
デジタル電源制御コントローラー向けHILS

SimPowerSystems™を用いて回路モデルを作成



制御対象の回路図

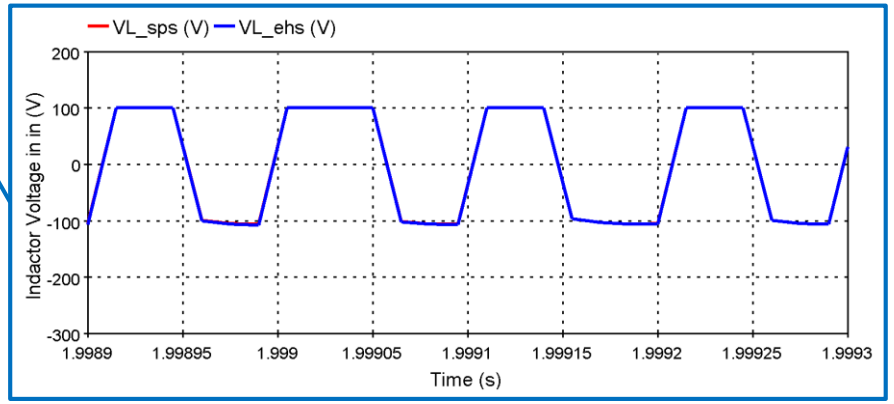




FPGA上に回路構成を再現

OPAL-RT社RT-LAB(リアルタイムシミュレータ)＋eHSソルバ

SimPowerSystems™を用いたシミュレーション結果(500ns,Tustin)とほぼ同じ精度でHILS環境の構築が可能



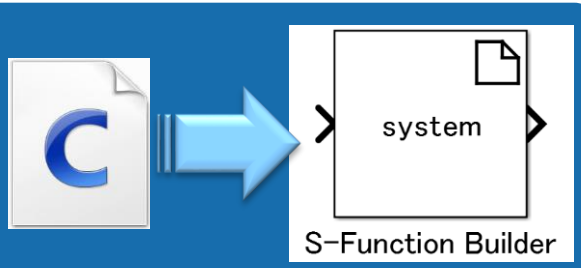
RT-LAB + eHS、SimPowerSystems™それぞれのシミュレーション結果
(※DC電圧:100V, PWMキャリア周波数:10KHz, Duty:50%)

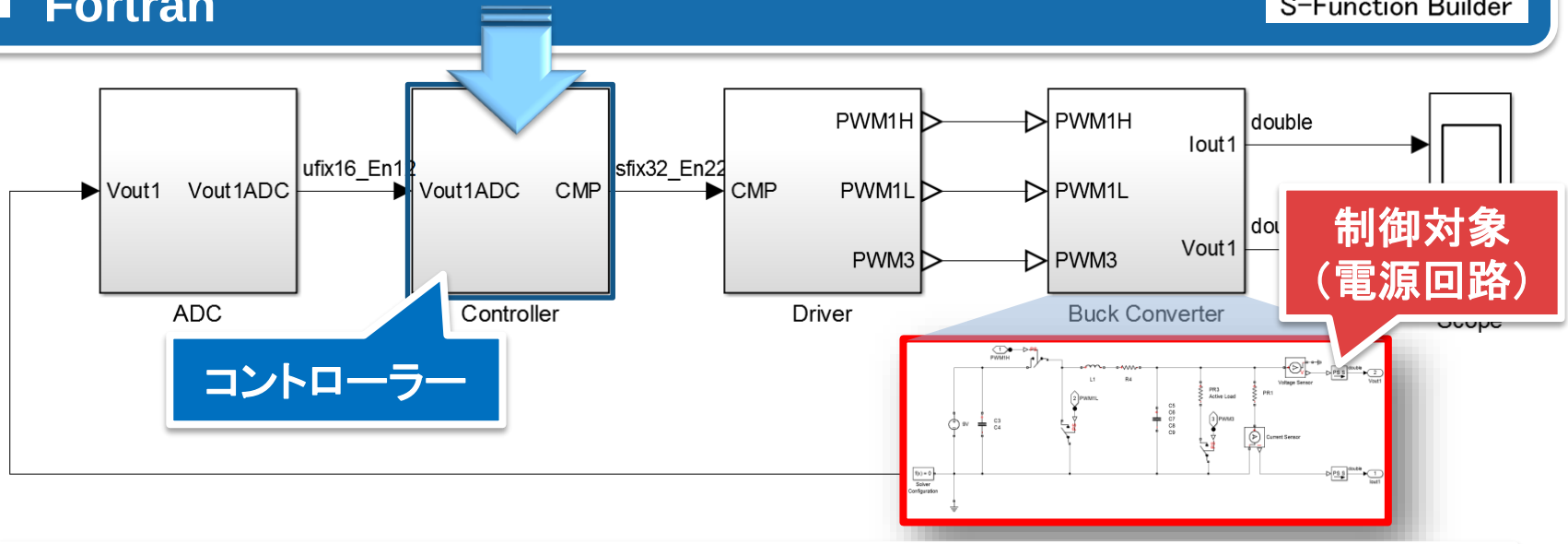
既存ソースコードのモデルへの活用

- コード資産をスムーズにSimulinkモデルで活用できます

既存ソースコード取り込み用ライブラリブロック

- MATLABスクリプト
- C言語/C++言語
- Fortran



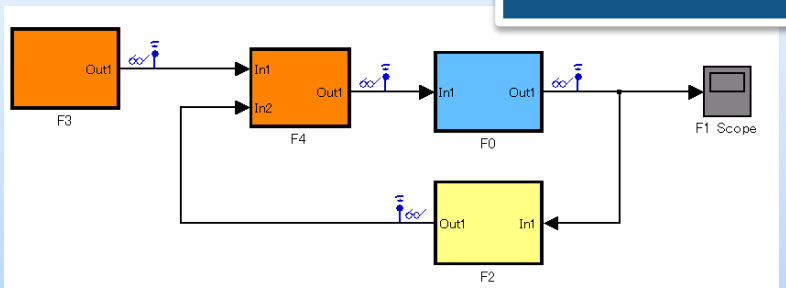


✓ ハンドコードのアルゴリズムをシステムレベルで動作検証
✓ 既存コードから徐々にモデルによる開発へシフトが可能

アナログ・ミックスシグナル回路設計へのモデルの活用
HDL VerifierによるSystemVerilog DPI生成

System Design

Simulink



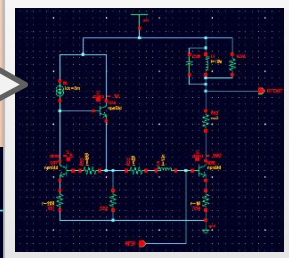
Cコード生成機能により
ビヘイビアモデルとしてエクスポート

Circuit Design

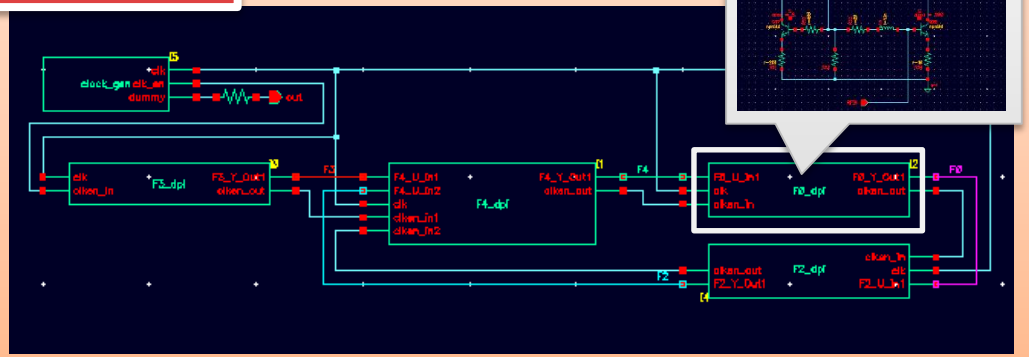
Cadence® virtuoso® AMS Designer

システムモデルの活用で詳細回路の検証&検証の効率化によるTATの削減

回路シミュレータ上で
詳細モデルの作り込み



- システムモデル活用で
  設計回路の早期システム検証
- システム等価検証に活用
- 複雑な信号で検証可能

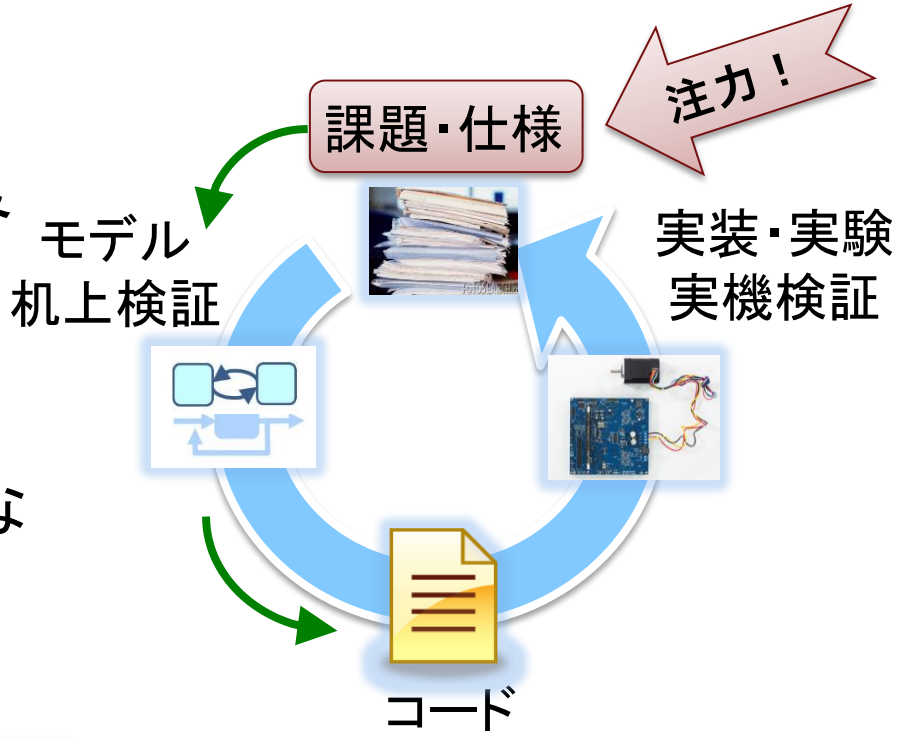


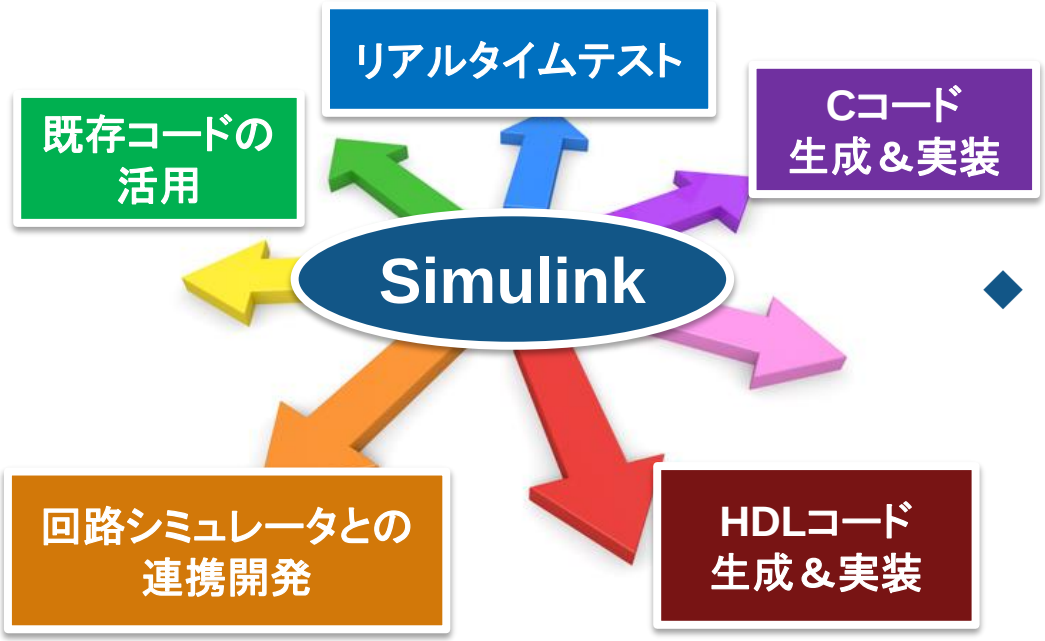
アジェンダ

- ソフトウェアへのアルゴリズム実装
- ハードウェアへのアルゴリズム実装
- 実機試験/他ツール連携ソリューション
- まとめ

まとめ

- モデルによるアルゴリズム開発＆コード生成ソリューションでシームレスな開発フローを実現
- 素早い修正ループでより本質的な開発業務にリソースをシフト





- モデル資産を柔軟に量産実装・試験まで有効活用

Accelerating the pace of engineering and science

ご清聴ありがとうございました